QTERM-R55 USER'S MANUAL
REVISION 6

QSI CORPORATION
2212 South West Temple #50
Salt Lake City, Utah 84115-2648
USA
Phone 801-466-8770
Fax 801-466-8792
Email info@qsicorp.com
Web www.qsicorp.com

M01-012-00 Rev 6

31725E0 - Printed in USA

Manual Updated 15 October 2009

FCC COMPLIANCE STATEMENT

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

Any modification to this device (including any changes to the recommended antenna configuration) that are not expressly approved by QSI could void the user's authority to operate this device.

FOREWORD

The QSI Corporation QTERM-R55 is a BASIC programmable data-entry terminal for industrial applications. The QTERM-R55 is available with several options; this manual discusses all versions and their operation.

The QTERM-R55 is a CE certified product. It has been assessed against the requirements of EN 50082-1: 1992, EN55022: 1987, and EN 60950 (including Amendments Numbers 1, 2 and 3). Based on conformity with these requirements, the QTERM-R55 is deemed in compliance with all applicable CE directives.

The sections of this manual are as follows:

Chapter 1. **Basic Operation.** This chapter helps you to quickly get acquainted with the QTERM-R55.

Chapter 2. **qaBASIC.** This chapter explains how to write a program in qaBASIC. It contains examples of how to use each of the commands, then an explanation of the command.

Chapter 3. **QTERM-R55 Hardware.** This chapter contains the specifications for the QTERM-R55 terminal. Included are dimensional drawings, interface specifications, connector pin assignments and environmental specifications.

Appendix A. **ASCII Chart.** This is a true 7-bit ASCII chart, along with mnemonic definitions.

Appendix B. **QTERM-R55 Character Chart.** This is a 256-character chart showing how the QTERM-R55 displays ASCII characters. The lower half is similar to, but not the same as, the true ASCII chart in Appendix A.

Appendix C. **qaBASIC Command Summary.** This is an abbreviated summary of the qaBASIC commands.

Appendix D. **Using the R55 Downloader.** This chapter describes how to use the R55 Downloader program.

CHAPTER 1.

BASIC OPERATION

There are only four steps required to communicate with the QTERM-R55:

- use *Power-On Setup* to set the display contrast, baud rate and data format
- connect to your host transmit, receive and ground lines
- apply power
- download application to the QTERM-R55

1.1 Power-On Setup

The *Power-On Setup* menu is used to configure the QTERM-R55's display contrast, baud rate and data format. This feature is also used to download new terminal firmware and to download the BASIC program file to the terminal. Normally, when power is applied to the unit, it will perform a few self-diagnostics and then check for a missing or bad program file. If the application file is bad or not present, the terminal will enter a download mode. If the program is present and uncorrupted, it is compiled and executed. An existing program file may be replaced with another one through the Power-On Setup routines. Alternatively, when a program ends normally (or due to an error), the QTERM-R55 will enter download mode.

To enter the *Power-On Setup* menu, follow these steps:

- Connect the power supply to the QTERM-R55.
- Hold down any three keys and apply power to the QTERM-R55 (you do not need to connect the transmit and receive lines). When the words **POWER ON SETUP** (POS) appear on the screen, release the keys.
- After a brief pause, a menu will appear. The asterisk indicates the currently selected menu item.
- The keys used for navigating the *Power-On Setup* menus are located in the upper right corner of the keypad (Figure 1-1 shows key locations for the 40-key keypad; the keys for the 24-key keypad are in similar locations with respect to the upper right corner of the keypad). The up arrow is hereafter referenced as the **POS Up** key. The other marked keys are similarly designated as **POS Down**, **POS Left**, **POS Right** and **POS Enter**.

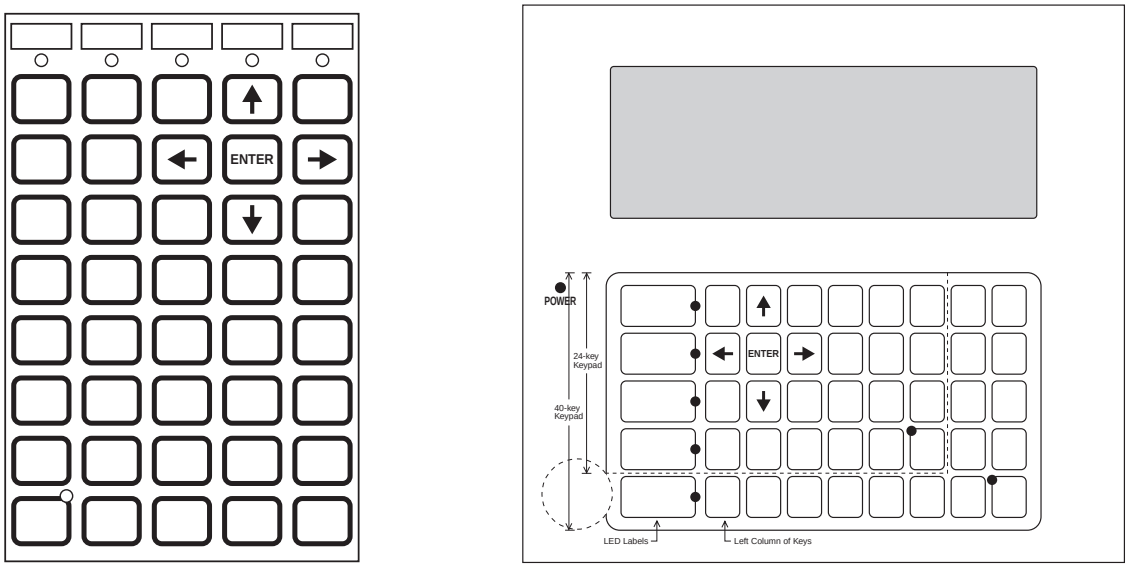


Figure 1-1. Key Names.

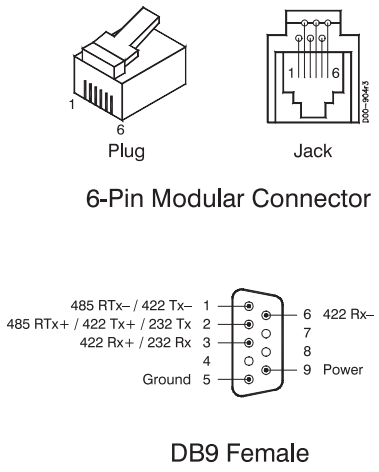


Figure 1-2. QTERM Connector Pin Assignments.

- Use the **POS Up**, **POS Down**, **POS Left** and **POS Right** keys to highlight the menu item for the parameter which you would like to change. Press the **POS Enter** key to select that item.
- The **System** menu allows adjustment of the display contrast and keyboard settings. When **Contrast** is selected the word CONTRAST will appear on the screen. The desired contrast is increased or decreased by pressing the **POS Up** or **POS Down** keys repeatedly. Press the **POS Enter** key to return to the POS menu.
- Keyboard settings such as Key Click, Key Repeat, Repeat Delay and Repeat Rate are accessed from the **Keypad** menu under the **System** menu. The Key Click and Key Repeat are toggled on and off by pressing the **POS Enter** key when **Click** or **Rpt** are selected. The Repeat Delay and Repeat Period may be incremented or decremented in 10 ms increments by pressing the **POS Up** or **POS Down** after **Rpt Dly** or **Rpt Prd** has been selected. When the desired value is displayed, press the **POS Enter** key to accept the value.
- Selecting **Back** from any menu returns to the previously displayed menu.
- The **COM Setup** menu is used to set the serial port baud rate and data format for the **Primary Port** or the **Secondary Port** (if the QTERM-R55 was purchased with the auxiliary serial port option). The baud rate is adjusted up and down with the **POS Up** and **POS Down** keys; press **POS Enter** when the desired baud rate is displayed. Select data format to adjust the number of data bits, parity and number of stop bits for the serial port. Selecting **Back/Save** from the menu saves these parameters and returns to the main POS menu.
- The **Clock** menu is used to set the Real Time Clock. If the QTERM-R55 was not purchased with the Real Time Clock option, **Clock** will not be displayed on the main POS menu.
- The **Defaults** menu restores all settings, including contrast, baud rate and data format, to their default values. No submenu exists for this item.
- The **Download** menu allows new terminal firmware and BASIC programs to be downloaded to the terminal. Select **Firmware** to upgrade the terminal firmware. The terminal will enter its bootloader routine and await receipt of the new firmware via the primary serial port. Selecting **Application** will enter a BASIC program download routine. The terminal is now ready to receive your BASIC program via the primary serial port. **Back** will return to the main POS menu without receiving any files.
- If the QTERM-R55 is configured with the 1.0 Mbyte flash memory option, the additional memory may be reserved for application storage (i.e. the qaBASIC program is stored in this space) or user file space. Choosing **Application** will display a menu to select how much memory is assigned to each of these tasks. Assign the memory by pressing **POS Left** and **POS Right**, then accept the settings by pressing **POS Enter**. The terminal will now enter the BASIC program download routine. The default partition (128 kbytes for application space and 640 kbytes for the file system) is usually appropriate unless your application is very large.

1.2 Connect the Communications Lines

The EIA-232 device has one transmit and one receive line, while the EIA-422 device has two transmit and two receive lines. The EIA-485 device has two bidirectional (transmit/receive) lines.

Table 3-2 in Chapter 3 shows the connector pin assignments for all versions. The receive and transmit directions shown in the table are relative to the QTERM-R55. Refer to this table to connect your host communications lines to the correct pins. Figure 1-2 shows the pin numbering of the 6-

pin modular connector (both jack and plug). The panel mount interface plug also uses this pinout.

(If you are using an IBM-style PC, you cannot connect an EIA-422 or EIA-485 QTERM-R55 directly to the computer's COM port; you must provide an interface device.)

1.3 Apply Power

Table 3-2 in Chapter 3 shows the pin assignments for the power and ground lines. Connect your DC power supply to the appropriate two pins.

WARNING: Power supplied to the QTERM-R55 must be from a SELV power source, and should have a current limit on its output of 5 Amperes. The supply to the QTERM-R55 must provide a minimum of 8 volts DC and be limited to a maximum of 32 volts DC. Limiting may be inherent to the supply or may be provided by supplementary overcurrent devices.

If the QTERM-R55 does not respond, or exhibits abnormal behavior on power up, disconnect power and contact QSI for technical support.

1.4 Running the R55 Downloader

See Appendix D. for information about the R55 Downloader for Windows™.

CHAPTER 2.

QABASIC

The QTERM-R55 is a user programmable operator interface terminal for industrial applications. The terminal is programmed using qaBASIC, a dialect of the common and simple BASIC programming language. The BASIC program can be written using any computer text editor and then downloaded into the terminal and stored in non-volatile memory. This program is executed after power is applied to the system. The program may be updated through the Power-On Setup menu; see section 1.1.

qaBASIC is a modified version of YABASIC, a BASIC compiler and runtime execution engine written by Marc Oliver Ulm. Much of this chapter was derived from the documentation that is included with the YABASIC distribution. YABASIC is distributed under the terms of the GNU Public License which allows modification and redistribution of the software provided that the same license applies to the modified version. Source code for the YABASIC modification may be obtained from QSI upon request. Contact QSI for details. Refer to the LICENSE file included on the distribution disk for more information on the GNU Public License.

The QTERM-R55 also uses uC/OS, the Real Time Kernel, by Jean Labrosse.

2.1 Syntax

2.1.1 Case

Commands can be entered in any case.

`input` is the same as **`INPUT`** and even as **`InPUt.`** This applies to every command in qaBASIC.

2.1.1.1 Variables

Variable names are case sensitive (i.e. types of variables: `a$` and `A$` are different) and can be of any length. There are two sorts of variables:

2.1.1.2 String Variables

String variables always have a dollar sign (`$`) as the last character of their names, e.g., `a$`, `b12$` or `VeryLongName$` and may contain strings of any length. String variables can contain any characters except the NULL character (ASCII 0).

2.1.1.3 Numerical Variables

`a`, `c3po` or `ThisIsAnEvenLongerName` are examples of numerical variables and can contain real numbers like -1.3, 2, 15.3e44 or 0.

Variables (with the exception of arrays) need not be declared; their initial values are "" (for string variables) and 0.0 (for numerical variables).

2.1.1.4 Multiple Commands on One Line

More than one command can appear on one line. The commands must be separated with colons.

2.2 Introduction to qaBASIC

Example 1:

```
REM this is the first R55 BASIC program
input "Enter two numbers:" a,b
print a,"+",b,"=",a+b
print "Please enter your Name:";
INPUT a$
print "Hello ",a$," !"
```

This program produces the following output (user input is displayed *`using this typeface`*):

```
Enter two numbers: 2 3
2+3=5
Please enter your Name: Bill
Hello Bill !
```

2.2.1 REM

The **REM** statement introduces comments. Everything after **REM** up to the end of the line is ignored. The **'** (apostrophe) is an abbreviated replacement for **REM**.

2.2.2 Input

This statement reads one or more variables from the user. The syntax for this command is

```
input [prompt] var1[,var2,...]
```

The optional [prompt] string after the **input** statement (`"Enter two numbers:"`) is printed on the terminal prior to reading any input. If the prompt string is omitted, the terminal uses the "?" character to prompt the user. Note that there is no semicolon or comma after this prompt string. If multiple variables are requested, they are listed after the prompt string, separated by commas.

The default input source is the keypad. This command may also be used to read input from a serial port or the non-volatile file store. See Serial Communication (section 2.8.4) and Nonvolatile Storage of Data (section 2.10) for details.

2.2.3 Print

The **print** statement writes all its arguments to the display. After writing its last argument, **print** advances the cursor to the next line (as in `print "Hello ",a$," !"`). To do this, the QTERM-R55 inserts a carriage return character (0x0D) and a linefeed character (0x0A) after the string. To avoid this behavior, place a semicolon after the last argument (as in `print "Please enter your Name:";`).

Note that print can be abbreviated with a single question mark (?). If you want to print (or input) at a specific location, refer to Printing and Controlling the Screen (section 2.9).

The default output destination is the display. This command may also be used to send output to a serial port or the non-volatile file store. See Serial Communication (section 2.8.4) and Nonvolatile Storage of Data (section 2.10) for details.

2.3 Arithmetic

Example 2:

```
print 1+2,2*3,4/2,2^3
print sin(1.0),cos(pi),tan(3)
print asin(0.5),acos(0.7)
print atan(2),atan(1,2)
print exp(1),log(2),log(euler)
print int(2.34),frac(2.34)
mod(11,4)
print min(2,3),max(2,3)
print sqrt(2)
```

This program produces the following output:

```
3 6 2 83
0.841471 -1 -0.142547
0.523599 0.795399
1.10715 0.463648
2.71828 0.693147 1
2 0.34
3
2 3
1.41421
```

2.3.1 Operations

qaBASIC has five arithmetic operators: `+` (addition), `-` (subtraction), **`*`** (multiplication), **`/`** (division) and **`^`** (power); they all behave as expected, as in Example 2.

Note that the power operator (`^`) handles fractional powers: `8^(1/3)` returns `2` as a result.

2.3.2 Functions

This section demonstrates and explains the arithmetic functions of qaBASIC.

2.3.2.1 Trigonometric Functions

Example 2 illustrates the six supported trigonometric functions.

qaBASIC can calculate sine, cosine, tangent and their inverses. All these functions expect their argument in radians; to facilitate the transformation from degrees to radians (`radian=degree*pi/180`), there is a predefined variable named **pi** (or **PI**) which has an initial value of 3.14159.

The **atan()** function has two forms. Called with a single argument (e.g. `atan(2)`) **atan()** returns a value between -pi/2 ... +pi/2. Called with two arguments (e.g. `atan(2,-1)`) **atan()** returns a value between -pi and +pi. (This can be useful when transforming from cartesian to polar coordinates).

2.3.2.2 Exponentiation

The **exp()** functions comes with its inverse: the **log()** function. **log()** and **exp()** operate with the base e

(2.71828) which comes as a predefined variable named **euler**. The exponentiation functions are illustrated by the second qaBASIC example.

2.3.2.3 Integer and Fractional Parts

The functions **int()** and **frac()** split their argument at the decimal point (see Example 2). **int** drops the fractional portion of the argument and returns an integer. **frac** returns just the fractional portion of the argument.

2.3.2.4 Remainder

To get the remainder of a division use the **mod()** function. As Example 2 shows, `mod(11,4)` produces `3`, because 11/4 = 2 with a remainder of 3.

NOTE: Arithmetic operations using large numbers may cause fractional inconsistencies in your answer. For example, 12000000 / 300000 may yield 40.00000001. This is inherent in floating point arithmetic.

2.3.2.5 Minimum and Maximum

The functions **min()** and **max()** return the lower and higher value of their two arguments respectively (see Example 2).

2.3.2.6 Square Root

The square root is calculated by the **sqrt()** function (see Example 2).

2.4 String Operations

BASIC has always been simple and strong in string processing. qaBASIC also maintains this feature.

Example 3:

```
a$="123456"
print len(a$)
print left$(a$,2),"-";
print mid$(a$,2,3),"-";
print right$(a$,3)
left$(a$,2)="abcd":print a$
print str$(12)
print str$(12.123455,"%08.5f")
print 2+val("23")
print val("e2")
instr("Hallo","al")
lower$("aBcD12fG")
ltrim$(" foo ")
print asc("e")
```

This program produces the following output:

```
6
12-234-456
ab3456
12
12.12346
25
0
2
abcd12fg
"foo "
101
```

2.4.1 Length of a String

The **len()** function returns the length of the string (see Example 3).

2.4.2 Extracting Parts of a String

There are three functions which return parts of a string:

```
left$(<string>,<length>)
right$(<string>,<length>)
mid$(<string>,<position>,<length>)
```

left$() returns the leftmost <length> characters of <string>. **right$()** returns the rightmost <length> characters of <string>, and **mid$()** cuts in the middle, returning <length> number of characters starting at <position> characters from the left end of <string>. The first character of a string is at position 1.

Furthermore, **left$()** and its associated functions can even be used to selectively change parts of a string by assigning a string to the function. Example 3 shows that only the two leftmost characters are changed (even though the string `"abcd"` contains four characters). The same can be done with **mid$()** or **right$()**.

2.4.3 Strings to Numbers (and Numbers to Strings)

The **str$ ( )** converts its numeric argument to a string (see conversion of `12` in Example 3).

Formatting of the number is optionally specified by a second argument (`"08.5f"` in Example 3). The second argument is essentially a format string as used by the `printf()` function in the C programming language. A subset of this function is supported by the QTERM-R55.

Format: `str$ (value, "%Flagfieldwidth.PrecisionArgument")`

Value		Value is the variable to be converted
Flag		OPTIONAL – One or more flags can be used to modify the result of the output, as follows
	–	Left adjustment of the output in the specified field width
	+	Number will be printed with a sign
	space	If the first character does not contain a sign, a space will be added
	0	Leading zeros will pad the field
	#	A decimal point will always be added. For g and G formats (see below), trailing zeros will be removed
Fieldwidth		REQUIRED – A number that specifies the minimum width of the field. The output will be printed in a field at least this wide, wider if necessary. The fewer characters than specified. Padding on the right will occur if left alignment (–) was used.
. (period)		OPTIONAL (required if precision is used) – Separates the Field-width from the Precision.
Precision		OPTIONAL – Maximum number of characters to be printed after the decimal point for e, E, and f output. For g and G, it represents the number of significant digits.
Argument		
f	f	Floating point ouput: (–)xxx.yyy The number of y digits is specified by the Precision 6 is the default precision, 0 will suppress the decimal point unless the # flag is used
	e, E	Floating point ouput: (–)xx.yyyye zz or (–)xx.yyyyE zz The number of y digits is specified by the Precision 6 is the default precision, 0 will suppress the decimal point
	g, G	Floating point ouput: %e or %E are used if the exponent is less than -4 or greater than or equal to the Precision, otherwise use %f. Trailing zeros and decimals are not printed.

More examples are listed in Table 2-0.

Further information can be found in any textbook on the C programming language.

The **val()** function converts its string argument to a number (see conversion of "23" in the Example 3). Note that "e2" is converted to 0 because this string is not a valid number.

Table 2-0. More Examples Using the str$ Function.

Print Statements	Output Produced
print "=" str$(12.12345,"%08.3f"); print "="	=0012.123=
print "=", str$(12.12345,"%8.2f"); print "="	= 12.12=
print "=", str$(12.12345,"%-6.2f"); print "="	=12.12 =

2.4.4 Finding Strings in Strings

The **instr()** function returns the position of its second string argument within the first. **instr()** returns zero if the string cannot be found.

instr("Hallo","al") in Example 3 returns 2 because "al" appears at position 2 within "Hallo". instr("Hallo","Al") returns 0, because "Al" is not contained in "Hallo" (the case doesn't match).

2.4.5 Changing the Case of Strings

lower$() and its counterpart **upper$()** convert their string argument to all lower or all upper case characters respectively.

lower$("aBcD12fG") returns "abcd12fg" as shown in Example 3.

2.4.6 Removing Spaces

ltrim$() and **rtrim$()** are two functions to remove leading or trailing spaces from a string. **trim$()** removes both leading and trailing spaces.

ltrim$(" foo ") returns "foo " (see Example 3) and rtrim$(" foo ") returns " foo". trim$(" foo ") returns “foo”.

2.4.7 The ASCII Character Set Functions

qaBASIC offers two functions to work with the ASCII character set.

`asc()` converts a specific character to its ASCII value. `print asc("e")` returns 101 as a result (see Example 3), because the character `"e"` has position `101` within the ASCII character set. Appendix A contains a complete 7-bit ASCII chart.

Likewise the function **`chr$()`** returns the ASCII character for a given position within the character set, e.g. `chr$(98)` returns `"b"`.

2.4.8 Escape Sequences

The most important non-printable characters can be constructed using escape sequences with the **** character: The sequence `\n` might be used instead of `chr$(10)` for the newline character.

Table 2-1 lists all escape sequences of qaBASIC (these are similar to the sequences used by the C-language).

Table 2-1. Escape Sequences for qaBASIC.

Escape Sequence	Resulting Character	ASCII Value
\n	newline	10
\t	tabulator	9
\b	vertical tabulator	11
\v	backspace	8
\r	carriage return	13
\f	form feed	12
\a	alert	7
\\	backslash	92
\‘	single quote	39
\”	double quote	34

These escape sequences are replaced within every pair of double quotes (**`""`**), i.e. within literal strings. User input read with the **`input`** statement is not affected in any way.

2.5 Conditions and Flow Control

Example 4:

```
input "Please enter a number" a
if (a>10) then
print "Hello ";
print "Your number is bigger than 10"
else
print "Byebye ";
print "Your number is less or equal 10"
endif
```

This program produces the following output:

```
Please enter a number 2
Byebye Your number is less or equal 10
```

Alternatively:

```
Please enter a number 11
Hello Your number is bigger than 10
```

Example 5:

```
input "Please enter a number" a
if a>10 and a<20 then
rem parentheses are optional
  print "bigger than 10 ";
  print "but less than 20"
fi
```

This program produces the following output:

```
Please enter a number 11
bigger than 10 but less than 20
```

Alternatively:

```
Please enter a number 10
```

Example 6:

```
label loop
? "Enter a string containing \"R55\""
input a$
if (instr(upper$(a$),"R55")<>0) then
  gosub thanx
else
  print "No, please try again !"
endif
goto loop
label thanx
print "Thanks a lot !"
return
```

This program produces the following output:

```
Enter a string containing "R55"
?thequickbrownfox
No, please try again !
Enter a string containing "R55"
?jumpedR55overthelazydog
Thanks a lot !
```

2.5.1 The `if` Statement

The **`if-then`** statement is necessary for making decisions. The syntax is as follows:

```
if [(]<condition>[)] then
   <instructions>
[else
   <alternative instructions>]
endif
```

As shown, the parentheses around <condition> are optional. The **`else`** <alternative instructions> is also optional. Note that **`endif`** can be written as **`fi`**.

<condition> is described below in Conditions (section 2.5.2). <instructions> and <alternative instructions> can be be any series of BASIC statements. If <condition> evaluates to TRUE, then <instructions> is executed. If <condition> evaluates to FALSE, <alternative instructions> (if included) is executed.

2.5.2 Conditions

Numbers or arithmetic expressions can be compared with the usual relational operators: **`=`** (equal), **`<>`** (not equal), **`<`** (less than), **`<=`** (less or equal), **`>`** (greater than) and **`>=`** (greater or equal).

Strings can be compared with the same set of operators, where characters are ordered according to the ASCII character set. For example, `("a"<"b")` is true because `"a"` precedes `"b"` within the ASCII character set. Likewise `("a"="b")` is false.

More than one comparison can be combined with parentheses **`()`** and these keywords: **`or`**, **`and`**, **`not`**. **`not`** has higher precedence than **`and`**, which in turn has higher precedence than **`or`** (in the same way as **`*`** precedes **`+`** within arithmetic expressions). This means that

```
not a>b or a==c and b>0
```

is the same as

```
((not a>b) or (a==c)) and (b>0)
```

Finally, the enclosing parentheses can be omitted, i.e. `if a<10 then ...` is a valid statement.

2.5.3 Marking Locations in a Program

The first line in the Example 6 program (`label loop`) is a **`label`**.

qaBASIC does not require line numbers, thus labels are necessary to mark a specific location within your program (however, see the following paragraph). You can compose labels out of letters and digits. The keyword **`label`** is required, and the label itself should be unique within your program.

qaBASIC allows for line numbering. This feature makes qaBASIC more compatible with traditional versions of BASIC. Line numbers are just special types of labels with the following properties:

- Line numbers can appear only at the beginning of a line.
- Not every line needs a number and line numbers need not be consecutive.

2.5.4 Jumping Around in a Program

A label by itself causes no special action. Only in conjunction with the **`goto`** statement (or **`gosub`** or **`restore`**) does a label have any function. If qaBASIC encounters a **`goto`** statement (in the Example 6 `goto loop`), then it searches for the matching label (`label loop`) and proceeds to execute at the position of the label.

Note that you can even leave (and enter) a for-next loop (see section 2.6) with goto.

Closely related to the **`goto`** command is the **`gosub`** command. If qaBASIC encounters a **`gosub`** statement, it searches for the matching label (`label thanx` in Example 6), and proceeds with execution at the position of the label until it finds a **`return`** statement. **`return`** makes qaBASIC return to the position of the original gosub and proceed from there.

Note that both **`goto`** and **`gosub`** can be used as **`on goto`** and **`on gosub`** (see on gosub, on goto (section 2.5.5).

2.5.5 on gosub, on goto

The **`on gosub`** statement is followed by a list of labels (`sorry,one,two,...`) in Example 9 (below). Depending on the value of the expression (`number+1`), the corresponding label in the list is chosen: for example, if `number+1` evaluates to 3, the third label (`three`) is selected and a **`gosub`** to this label is performed.

A **`gosub`** is always performed, regardless of the value of the expression. More specifically, if `number+1` gives anything less or equal to 1, then the first label (`sorry`) is chosen. If `number+1` evaluates to anything greater or equal to the number of elements in the list (which is 7 in Example 9), then the last label (`sorry`) is chosen. Therefore, the

label `sorry` is chosen whenever the program cannot convert the given number.

Note that the **on** construct can also be used as **on goto**.

2.5.6 End of Program

The **end** statement ends your program immediately (see Example 9 below). Note that this will cause the QTERM-R55 to enter download mode and wait for a new application program to be sent via the serial port.

2.6 Loops

Example 7:

```
input "Please enter a word" a$
for a=len(a$) to 1 step -1
print mid$(a$,a,1);
next a print " is ",a$," reversed !"
```

If you try this program, you will get this output:

```
Please enter a word: hello
olleh is hello reversed !
```

In the above program, everything from `for` to `next` is repeated, while the variable (`a`) goes from its initial value `len(a$)` to its final value `1`.

Note the **step** clause. The number after **step** (here: `-1`) is added to `a` after every repetition. The **step** clause makes `a` go down with every iteration. If the step clause 3 is omitted, `step 1` is assumed.

Within the **for-next** loop above the string functions, **len()** and **mid$()** are applied (see section 2.4).

2.7 Data and Arrays

Occassionally the need arises to supply a program with initial data. Example 8 program converts numbers to strings.

Example 8:

```
restore names
read maxnum
dim names$(maxnum)
for a=1 to maxnum:read names$(a):next a
label loop
input "Please enter a number: " num
num=int(num)
if (num>=1 and num<=maxnum) then
  print num,"=",names$(num)
  goto loop
endif
print "Sorry, can't convert ",num
label names
data 9,"one","two","three"
data "four", "five","six"
data "seven","eight","nine"
```

The following program (Example 9) produces similar output but is written using totally different language constructs.

Example 9:

```
label loop
input "Please enter a number: " number
number=int(number)
on number+1 gosub sorry,one,two,three,
four,five,sorry
goto loop
label sorry
print "Sorry, can't convert ",number:end
label one:print "1=one":return
label two:print "2=two":return
label three:print "3=three":return
label four:print "4=four":return
label five:print "5=five":return
```

These programs produces the following output:

```
Please enter a number: 2
2=two
Please enter a number: 3
3=three
Please enter a number: 12
Sorry, can't convert 12
```

2.7.1 Reading Data

Example 8 program converts numbers to their textual representation. For this purpose it needs to know the numbers from 1 to 9 as text. This information is stored with the **data** lines at the bottom of the program.

The program uses the **read** command to get one piece of data after the other.

If you want to deviate from the linear ordering while reading the **data** statements, you may use the **restore** statement. In the example above, `restore names` makes sure that the next **read** statement reads its data after the label **names**.

2.7.2 Arrays

In Example 8, the words `"one" ... "nine"` are stored within the string array `names$()`.

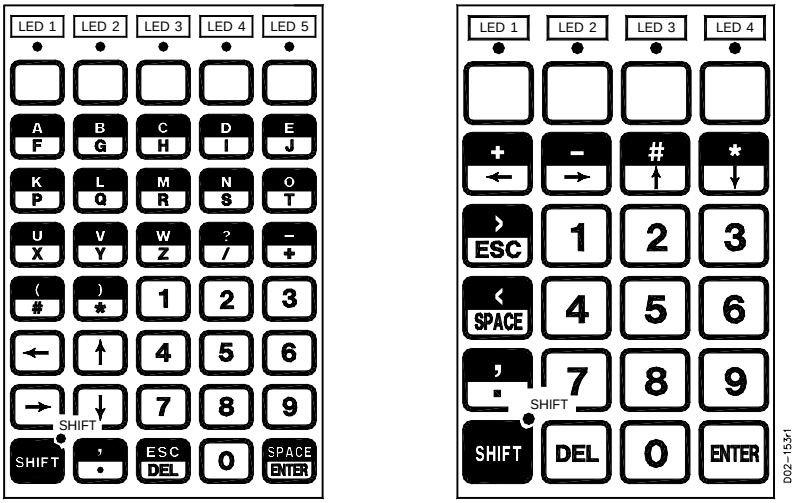


Figure 2-1. QTERM-R55 Keypad LEDs.

Arrays can be used to process large quantities of data. There are numerical arrays as well as string arrays. Both need to be declared prior to their first use. This is necessary because qaBASIC needs to know how much memory has to be allocated for the array.

The **dim** keyword is used to declare arrays. Example 8 uses `dim names$(maxnum)` to declare a string array. Another example would be `dim numbers(200)` to create a numerical array with 200 elements numbered 0 to 199.

More complex tasks may even require multidimensional arrays with more than one index. `dim matrix(10,10)` defines a two dimensional array. Up to 10-dimensional arrays are supported by qaBASIC.

2.8 Interaction with System Hardware

Example 10:

```
print date$," ",time$
cursor blink
cursor at(2,3)
a$="12345":a=4
send #COM1 a$
send #COM1 a
a$ = recv$ #COM1
a = recv #COM1
print #COM1 a$,a
led 3 on
led 4 off
backlight on
bell
pause 10
```

This program produces the following output (phrases in parentheses describes actions taken by the QTERM-R55, not output to the display):

```
5 28 08 98 13:51.53
(cursor set to blink)
(cursor moves to column 2, row 3)
(str$ "12345" sent out COM1)
(num 4 sent out COM1)
(COM1 input stored into str$ a$)
(COM1 input stored into num a)
(string in str$ a$ and number in num a
 is sent out COM1)
(LED 3 is turned on)
(LED 4 is turned off)
(backlight is turned on)
(a speaker beep is made)
(execution stops for 10 seconds)
```

2.8.1 Date and Time

NOTE: The **date$** and **time$** commands are only supported for QTERM-R55 terminals with the optional real-time clock installed.

The **date$** output string has four fields (see Example 10): `5` is the day of the week (1-7, 1 is Sunday, 7 is Saturday), `28` is the day of the month (01-31), `08` is the month (01-12) and `98` is the year.

The **time$** output string has three fields: 13 is the hour (00-23), 51 is the minute (00-59) and 53 is the second (00-59).

All fields of **date$** and **time$** (except the last field within **time$**) are fixed length; thus it is easy to extract fields with the **mid$** function [see Extracting Parts of a String (section 2.4.2)].

2.8.2 Cursor Appearance

The following **cursor** commands control the appearance of the cursor:

```
cursor normal
cursor blink
cursor off
```

Normal is the standard underscore cursor (default). The cursor may be positioned with the **at()** command as follows:

```
cursor at(X,Y)
```

where X and Y are the column and row of the desired cursor location, respectively. The **at()** command can also be used with the print command (see section 2.9.1).

These commands are illustrated in Example 10.

2.8.3 Peek

The **peek()** function is handy for obtaining certain information about the QTERM-R55 hardware. This command can be used to get the number of character rows and columns on the QTERM-R55 display, the number of keys on the keypad and the current cursor position (X or Y). Peeking at *COM1* or *COM2* returns the number of characters available in the receive buffer for that interface. Peeking at *ostick* returns the number of operating system time ticks since the unit was powered on. The time interval between ticks is 20 milliseconds. The syntax is:

```
peek(<"keyword">)
```

where <keyword> is one of the following:

```
columns
rows
numkeys
cursorX
cursorY
COM1
COM2
ostick
```

The numeric return value contains the requested information.

2.8.4 Serial Communication

The program may send characters or strings to the serial port with these commands:

```
send #COM1 string$
send #COM1 num
string$ = recv$ #COM1
num = recv #COM1
```

In a string context, **send** outputs string string$ to the port. In a numeric context, **send** outputs a character whose ASCII value is contained in num (truncated to an integer, modulo 255).

In a string context, **recv** will grab everything in the serial input buffer (up to 100 characters) and put it into string string$. If no characters are available, this function returns an empty string.

NOTE: As explained in section 2.1.1.2, string variables may not contain NULL characters. Therefore, if the serial stream might contain NULL characters, **recv** must be used in a numeric context.

In a numeric context, **recv** pulls one character from the serial port and puts its ASCII value into variable num. If no character is available, this function returns the value 256.

print and **input** also work with the serial port by inserting the serial port identifier:

```
print #COM1 string$,num
input #COM1 a$
```

Example 10 illustrates the serial communications commands.

Programming serial communications for the EIA-485 interface requires special attention. See section 3.4.3 for more information.

2.8.5 Keypad LEDs

The LEDs may be turned on and off with the following commands:

```
led <lednum> on
led <lednum> off
```

<lednum> is an expression yielding an integer from 1 to 6. See Figure 2-1.

Example 10 program illustrates the **led** commands.

2.8.6 Display Contrast

The display contrast may be adjusted with the **contrast** commands (see Example 10):

```
contrast up
contrast down
```

The new contrast setting persists until the terminal is powered down. To permanently adjust the contrast of the display, use the Power-On Setup facility (see section 1.1).

2.8.7 Display Backlight

The display backlight may be turned on and off with the **backlight** commands (see Example 10):

```
backlight on
backlight off
```

2.8.8 Key Click

If enabled, the Key Click feature causes the terminal speaker to emit a short beep whenever a key on the keypad is pressed. This feature may be turned on and off with the following commands:

```
keyclick on
keyclick off
```

The Key Click setting persists until the terminal is powered down. The default setting (when the terminal is powered on) can be selected in the Power-On Setup facility (see section 1.1).

2.8.9 Keypad Backlight

If the QTERM-R55 was purchased with the keypad backlight option, the keypad backlight may be turned on and off with the keypad commands:

```
keyback on
keyback off
```

2.8.10 Speaker

The internal speaker beeps in response to either of the following commands (see Example 10):

```
beep
bell
```

2.8.11 Pause Execution

The following identical commands delay program execution (see Example 7):

```
pause <time>
wait <time>
```

`<time>` is the amount of time execution is delayed in seconds.

2.8.12 Defining the Keyboard

Input from the keyboard can be done in two ways. The first method is to check periodically if there has been a key pressed by using the **iskeypressed$** string function. If there has been a key pressed, the resulting string from this function will be the value of the key that was pressed. Otherwise, the string will be empty. Here is an example of how this function can be used.

```
inChar$ = iskeypressed$
if (len(inChar$) <> 0) then
  send #COM1 inChar$
endif
```

The other method of getting input from the keypad is to use the **inkey$** string function. This function will stop execu

tion of the BASIC program until the user of the terminal presses a key on the keypad. When he does, the value of the key is returned as the result.

```
print inkey$
```

The keyboard has a default definition for characters that are returned to the input functions. This definition contains the characters that are printed on the standard legend; see Figure 2-1. However, as the QTERM-R55 legends are completely customizable, a qaBASIC facility has been created to allow for redefinition of the characters that are returned by input functions when keys are pressed. The command to accomplish this is described below.

```
keydef[release][shift] <col>,<row>,<string$>
```

This command redefines the key at location (col, row), where col and row are the one-based column and row of the key on the keypad. For example, the upper left key on

the 40-key keypad is key 1,1, and the lower right key is key 5,8. The **string$** is sent to the BASIC input functions when the key is pressed. The optional release and shift modifiers change the definition for a key release event and/or a shifted key event.

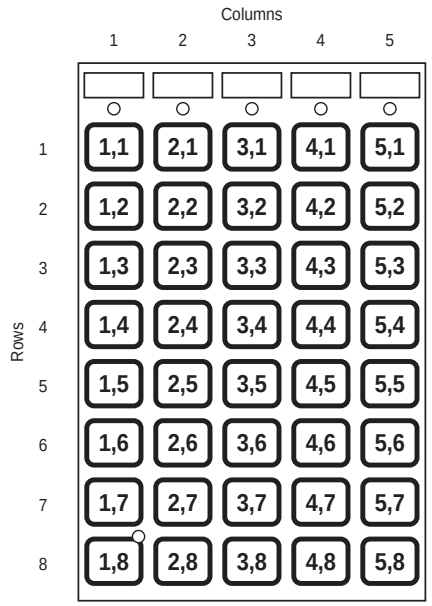


Figure 2-2. Key Locations

Key release events can be enabled with the **keyrelease on** command. The **keyrelease off** command disables them. The default is disabled.

2.8.13 Shift State

The state of the shift key (on or off) may be altered with the **shiftstate** commands:

```
shiftstate on
shiftstate off
```

The shift state is toggled automatically when the shift key is pressed (See Figure 2-1). When the shift state is on, the shifted key definition is used to determine the key input (seeDefining the Keyboard (section 2.8.12).

2.9 Printing and Controlling the Screen

2.9.1 Printing Output

Printing output at specific locations is helpful for interactive programs.

Example 11:

```
clear screen
print at(4,0) "1 -- Setup"
print at(4,1) "2 -- Save"
print at(4,2) "3 -- Quit"
input at(0,3) "Your choice: " a$
```

This program produces a screen resembling the following layout:

```
1 -- Setup
2 -- Save
3 -- Quit
Your choice:
```

The **clear screen** command erases the screen.

The **at()** clause in print or input statements allows you to move to any location (specified by the two arguments of the **at()** clause) on your screen. This is illustrated by Example 8. Note that **at()** can be written as **@()** also, and that the row and column addresses are zero-based (i.e. the first row is row 0, etc.).

2.9.2 Autowrap and Autoscroll

autowrap and **autoscroll** are screen display commands that allow the qaBASIC program to control how the display responds when printing and reaching the end of a line or the bottom of the screen. If the autowrap is on when the cursor reaches the end of the screen, then the cursor will be placed at the beginning of the next line (unless the line is the bottom line and autoscroll is off, in which case the cursor will be placed at the beginning of the bottom line). If autoscroll is on when the cursor reaches the bottom line of the display and receives a newline character, it will scroll the screen up (leaving the cursor in the same horizontal position).

The following syntax is used for autowrap and autoscroll statements.

```
autowrap on
autowrap off
autoscroll on
autoscroll off
```

2.10 Nonvolatile Storage of Data

The following commands were added to allow for a limited "file system." Although one may read and append, a file can't be overwritten because of the nature of flash memory. The entire file space must be erased in order to rewrite a file. The command to erase is given as a system command (described below). Up to nine files may be used for storage of data.

open #<1-9> [, "r" or "a"] opens a file for reading ("r") or appending ("a" -- append is default). Example:

```
open #1, "r"
```

print #<1-9> <str$>** prints **str$ (writes) to the end of a currently open file.

input #<1-9> <str$> reads from a currently open file until it reaches a newline character.

eof (#<1-9>) determines if the end of a file has been reached when reading.

close #<1-9> closes a currently open file.

system ("erase filespace") or **system$ ("erase filespace")** erases the filespace.

system ("avail filespace") or **system$ ("avail filespace")** is a pessimistic indication of how much free area exists for writing more data.

seek #<1-9> to line <expression> allows random access to an open file in the filespace. This command will evaluate **<expression>** to a positive integer *n* (dropping fractions) and will position a file pointer at the beginning of the *nth* line in the specified file. Subsequent **input** command will read the file at this location. A line is defined as any sequence of bytes ending in the newline character.

seek #<1-9> <expression> lines is similar to the **seek** command above, except that it positions the file pointer relative to the current file pointer location rather than to an absolute line number. This command evaluates **<expression>** to a positive or negative integer *n* (dropping fractions) and positions the file pointer to the beginning of the *nth* following line (for positive n) relative to the current file pointer location in the specified file. Subsequent **input** command will read the file at this location.

Print command can only be used to write to a file if it has been opened for appending and the current file pointer location is at the end of the file.

Note that the **print** and **input** commands automatically position the file pointer to the beginning of the following line after reading/writing the file.

2.11 Error Handling

There are two types of error handling statements that can be used:

```
on error goint <label>

on lowbat goint <label>
```

The first statement sets the terminal so that if it encounters a non-fatal error, it will begin execution of a special subroutine starting at the location given by the label rather than sending the error description to the primary serial port and aborting the program. Any subsequent errors encountered while handling that error, however, will not call the error handler but a description will be sent to the primary serial port. The error that caused the interrupt may be found in the error handling routine by using the command **peek$("error")**. The resulting string of this function will contain the error information.

The lowbat handling routine will also call a subroutine starting at the location given by the label, assuming the unit is designed for battery operation. If there is no error handler, and the unit's battery is getting low, it will beep periodically instead.

IMPORTANT NOTE: While in these interrupt handlers, one should not attempt to perform any type of input processing from the keypad or communications port that suspends program flow. Interrupt handlers should be simple and quick and should return with a gosub at the end of the routine (subroutine calls are allowed inside the handler and will be handled appropriately).

CHAPTER 3.

QTERM-R55 HARDWARE

3.1 QTERM-R55 Handheld Terminal

The dimensions of the QTERM-R55 handheld terminal are shown in Figure 3-1. This figure also shows the standard keypad legend for the keys on the 40-key keypad.

The housing is made from impact-resistant ABS and is colored black. The housing is not waterproof, but it can be subjected to moderate rain or splash without harm.

The QTERM-R55 handheld terminal uses a 6-pin modular jack, exiting from the bottom of the housing. If desired, the modular jack can easily be switched to exit from the top of the QTERM-R55 housing.

The pin assignments for the connectors for each of the available interfaces are shown in Table 3-2. Figure 3-2 shows the pin numbering for the connectors used on the QTERM-R55. For reference, Table 3-1 shows the pin assignments used by the COM ports on PC-style computers for both DB25 and DB9 connectors.

3.2 QTERM-R55 Panel-Mount Terminal

The panel-mount QTERM-R55 is mounted directly onto your instrument or enclosure. Figure 3-3 shows the dimensions of the panel-mount QTERM-R55 along with the standard keypad legend for the 24-key keypad.

The QTERM-R55 is mounted to the panel by placing the terminal into the panel cutout and attaching the rear shell behind the panel with the four self-tapping screws. A mounting gasket is provided in order to seal the front edge of the terminal. Figure 3-4 shows the assembly diagram along with the panel cutout dimensions required for mounting.

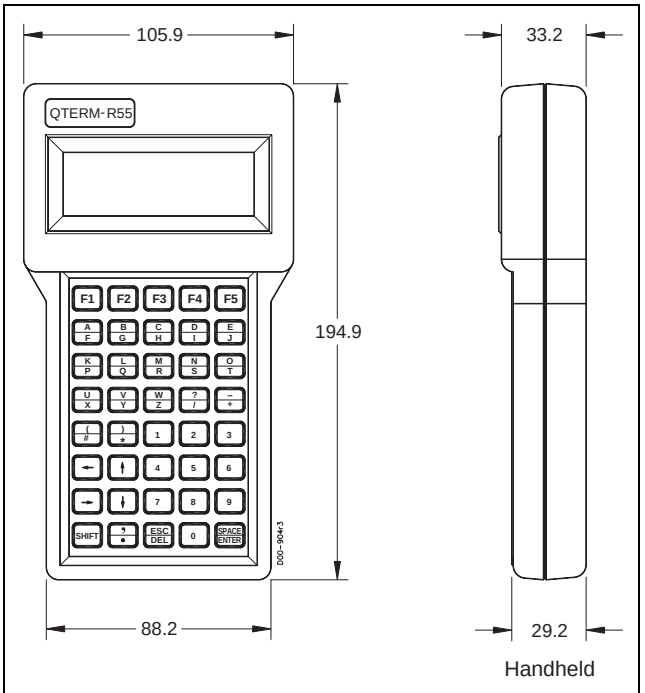


Figure 3-1. Dimensions of the QTERM-R55 Handheld Version.

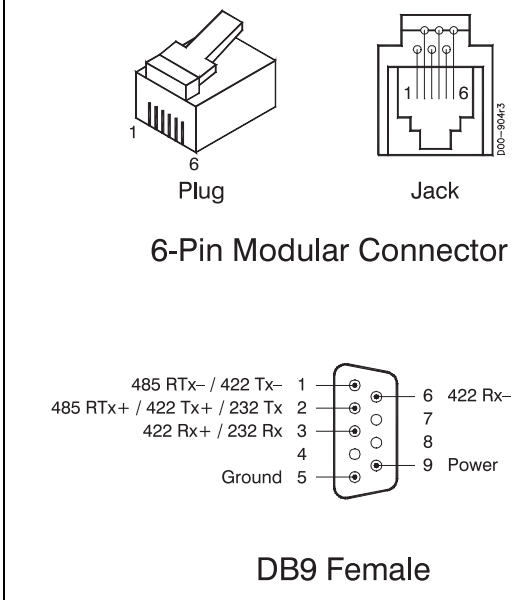


Figure 3-2. QTERM Connector Pin Assignments.

Table 3-1. Pin Assignments for PC-Style COM Ports.

COM Ports with Male DB25 Connectors		COM Ports with Male DB9 Connectors	
Pin	**Function**	**Pin**	**Function**
2	PC transmit	1	CD *
3	PC receive	2	PC receive
4	RTS *	3	PC transmit
5	CTS *	4	DTR *
6	DSR *	5	ground
7	ground	6	DSR *
8	CD *	7	RTS *
20	DTR *	8	CTS *
22	RI *	9	RI *

*These lines normally can be left unconnected. Some PCs may require that one or more of them be pulled to 5 volts through a pullup resistor (about 300 ohms)

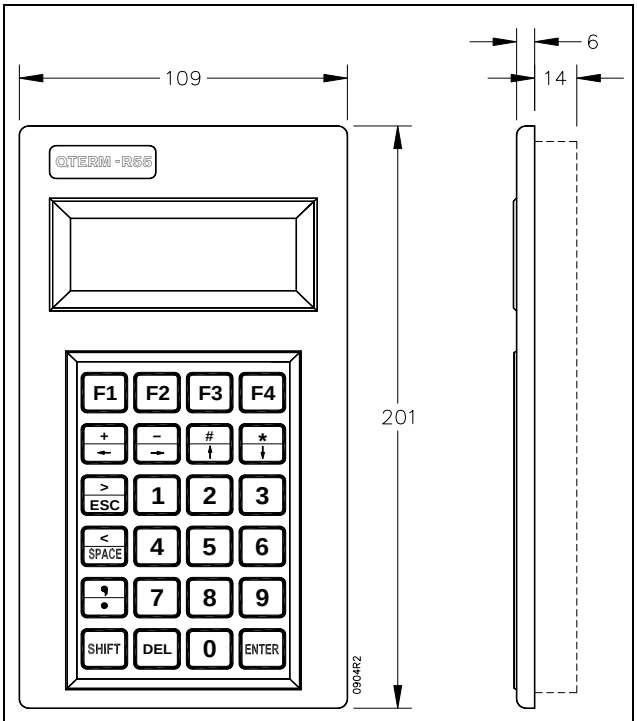


Figure 3-3. Dimensions of the QTERM-R55 Panel Mount Version.

Connection to the panel-mount QTERM-R55 is made via a 6-pin 2 mm pitch header which is located on the PCB as shown in Figure 3-5 (standard output).

3.3 QTERM-R55 Large Character Panel-Mount Terminal

The dimensions of the QTERM-R55 large character panel-mount terminal are shown in Figure 3-6. This terminal is available with a 24-key or a 40-key keypad.

The housing is made from rugged glass-fiber filled nylon plastic and is colored black. The housing is not waterproof, but it can be subjected to moderate rain or splash without harm.

The QTERM-R55 is mounted directly onto your instrument or enclosure panel. Figure 3-7 shows the mounting cutout required to install the terminal.The QTERM-R55 is mounted to the panel via four angle brackets that attach to the terminal with self-tapping screws that insert from the rear of the bracket and screw into the bosses on the QTERM-R55 enclosure. These brackets are designed so they can be attached before inserting the terminal into the panel. Once the brackets have been installed, the terminal is inserted into the panel and the brackets are slid outward. Then the screws are tightened and the brackets hold the terminal in place. Screws and brackets are included with the QTERM-R55; these are usable with panels up to about 10 mm thick. A gasket is provided to seal the outside edges of the terminal where it presses against the panel. Connection to the large character panel-mount QTERM-R55 is made via the DB9 connector which is accessible on the back of the terminal. The connector pin numbering is shown in Table 3-2.

Table 3-2. QTERM Pin Assignments

Handheld 6-pin Modular	6 x 1 2 mm	DB9 Female	QTERM EIA-232/ 5VB Function	QTERM EIA-422 Function	QTERM EIA-485 Function
1	1	3	receive	+receive	RTx-
2	2	6	RTS*	-receive	RTx+
3	3	2	transmit	+transmit	no connect
4	4	1	CTS*	-transmit	no connect
5	5	9	power	power	power
6	6	5	ground	ground	ground
*if the hardware flow control option is enabled					

3.4 Interfaces

3.4.1 EIA-232 Interface

With proper cables and good grounding, the EIA-232 interface on the QTERM-R55 can communicate up to about 15 meters.

Handshaking between the host computer and the QTERM-R55 is optional. This is done using software flow control (XON/XOFF) or the EIA-232 modem-control lines (such as RTS and CTS). Note the non-standard pinout for the RTS and CTS lines on the DB9 connector.

3.4.2 EIA-422 Interface

With proper cables and grounding, the EIA-422 interface can communicate up to a distance of about 1,000 meters. The EIA-422 version of the QTERM-R55 uses four communication lines and two power lines.

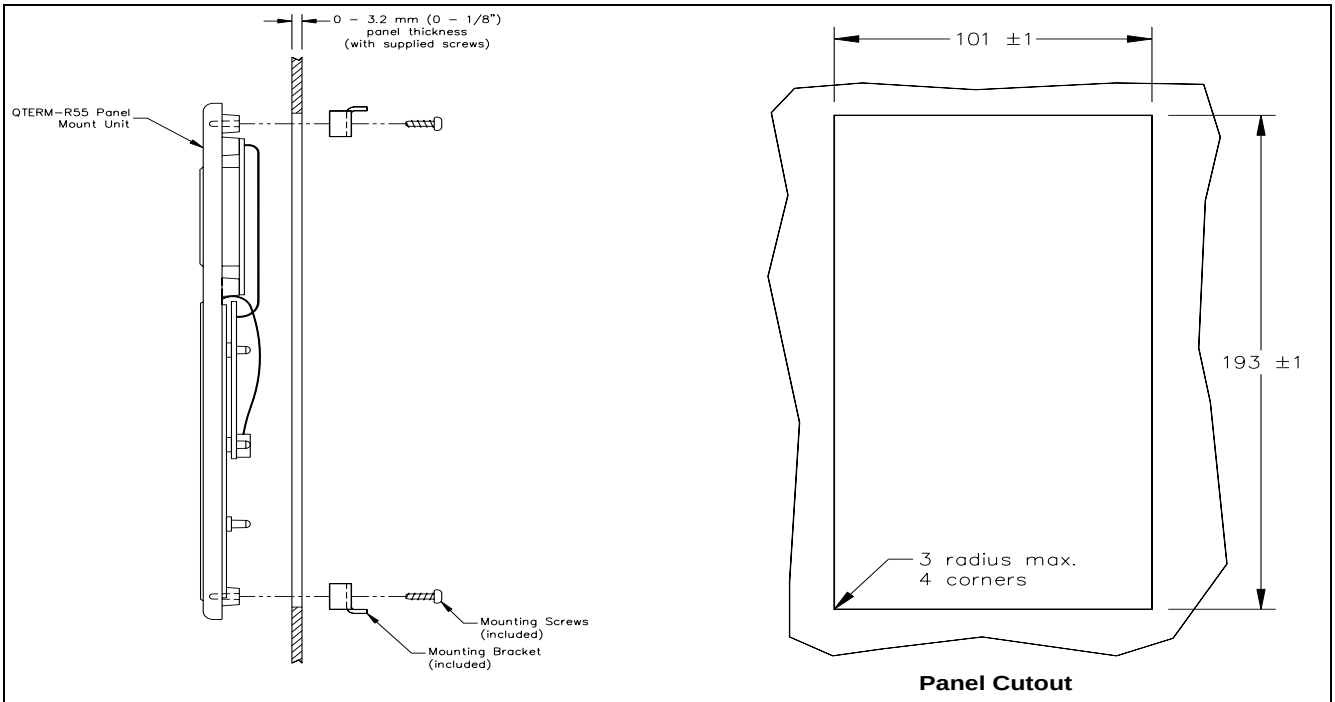
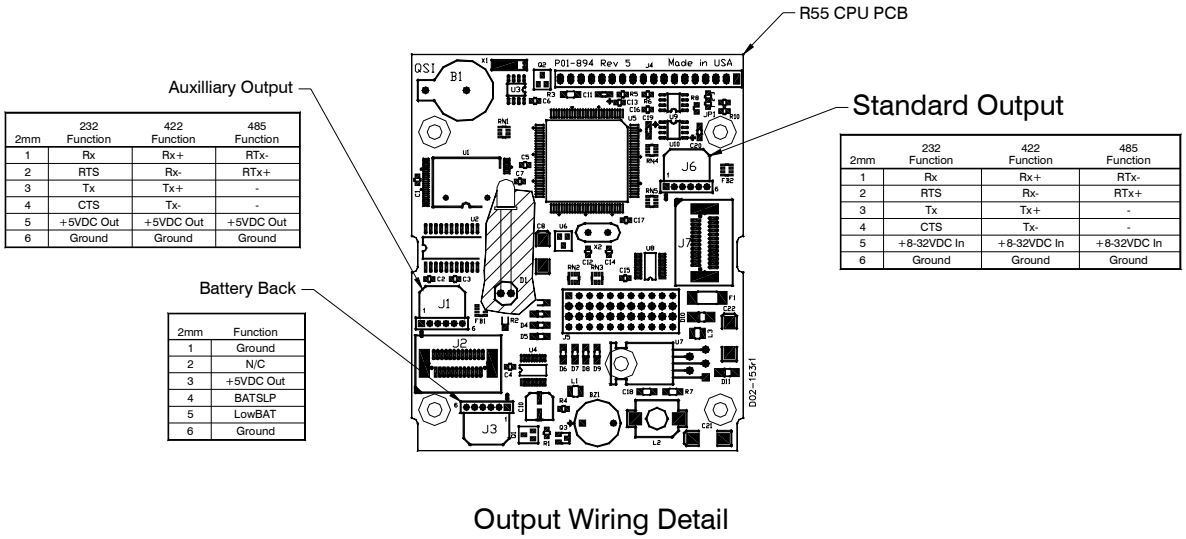


Figure 3-4. QTERM-R55 Assembly and Panel Cutout Dimensions.



2mm	232 Function	422 Function	485 Function
1	Rx	Rx+	RTx-
2	RTS	Rx-	RTx+
3	Tx	Tx+	-
4	CTS	Tx-	-
5	+5VDC Out	+5VDC Out	+5VDC Out
6	Ground	Ground	Ground

2mm	232 Function	422 Function	485 Function
1	Rx	Rx+	RTx-
2	RTS	Rx-	RTx+
3	Tx	Tx+	-
4	CTS	Tx-	-
5	+8-32VDC In	+8-32VDC In	+8-32VDC In
6	Ground	Ground	Ground

2mm	Function
1	Ground
2	N/C
3	+5VDC Out
4	BATSLP
5	LowBAT
6	Ground

Figure 3-5. QTERM-R55 Panel Mount Output Wiring

3.4.3 EIA-485 Interface

The EIA-485 interface is similar to EIA-422, but was designed as a half-duplex multidrop interface. With proper cabling and termination, up to 32 EIA-485 devices can be connected across a cable of up to 1000 meters in length. The EIA-485 interface on the QTERM-R55 uses two communication lines and two power lines.

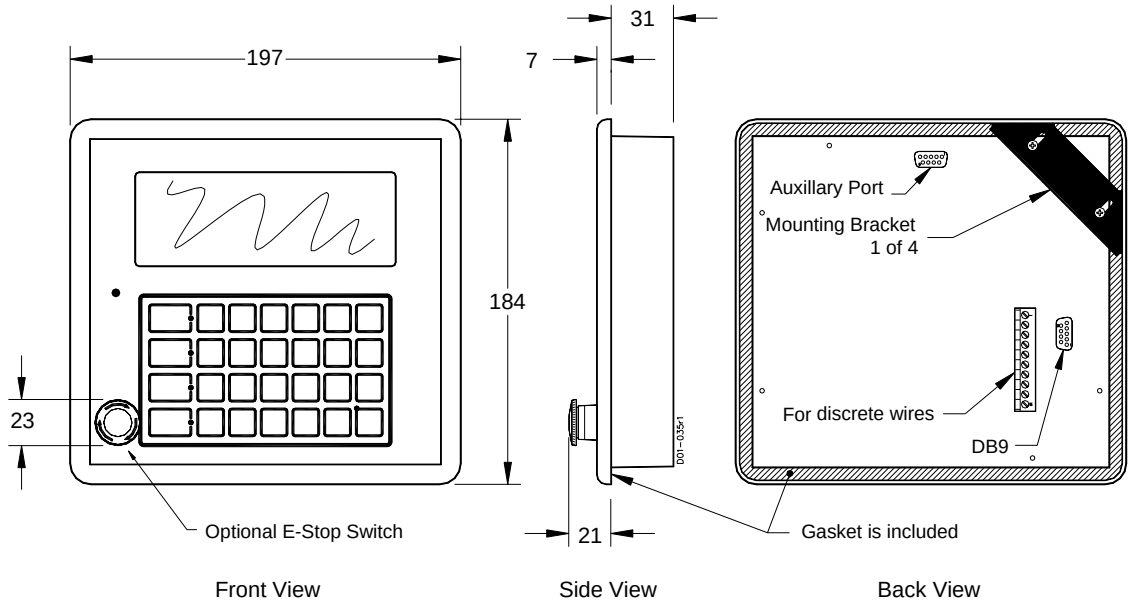


Figure 3-6. Dimensions of the QTERM-R55 Large Character Panel-Mount Terminal.

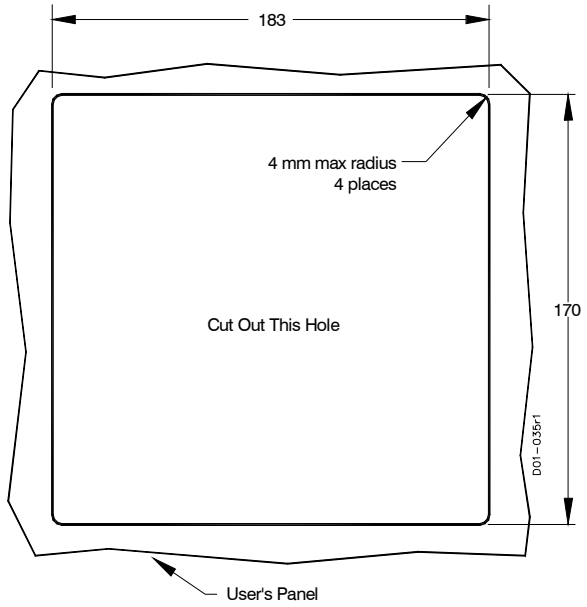


Figure 3-7. Cutout Dimensions for the QTERM-R55 Large Character Panel-Mount Terminal.

The EIA-485 interface uses intelligent electronics to control the direction of the interface (transmit vs. receive). The interface is held in receive mode until the user application transmits data. The circuit senses the beginning of the transmission and immediately switches to transmit mode until the transmission is ended. At that time, the interface reverts to receive mode until another transmission is requested. All of this is handled by the interface electronics, and is completely transparent to the user application. This can simplify the task of writing software for the EIA-485 interface, because the application does not need to directly control the direction of the interface.

The QTERM-R55 operating system does not specify or enforce any protocol or link layer functions on the EIA-485 interface. This must be provided by the user application. Most EIA-485 networks use some form of master-slave protocol to control network traffic. This is accomplished in software by adhering to a few simple rules:

1) Slave devices must never initiate a transmission unless they are queried by the master. The slave then needs to respond within a certain time period.

2) Master devices may initiate transmissions to slaves, but after doing so the master should wait (and not transmit again) until the slave has responded or the timeout period for the slave response has expired.

3) If broadcast queries (which require a response from multiple slaves) must be supported, the slaves should respond sequentially or otherwise resolve the response traffic to avoid contention.

3.5 LCD Display

The QTERM-R55 display is a 4-line by 20-character supertwist LCD unit. The entire 128-byte ASCII character set can be displayed; also, many Greek letters, katakana characters, non-English alphabetic characters and math symbols can also be displayed.

Appendix B. is a chart which shows how the QTERM-R55 will display each 8-bit value. Note that the ASCII portion of the chart (the first 128 characters) is similar, but not identical, to the true ASCII chart shown in Appendix A.

If you ordered the backlight, the qaBASIC commands listed in Chapter 2 will allow you to turn the backlight on and off. Without the backlight, these commands have no effect.

3.6 Keypad

The QTERM-R55 is available with a 24-key or a 40-key keypad. The *Power-On Setup* menus allow you to control both key click (on or off) and key repeat (on or off, delay and rate).

3.7 Other Options

3.7.1 Speaker Option

The QTERM-R55 includes an audio speaker which is used for key beeps and for beeping in response to a beep or bell command.

3.7.2 Real-Time Clock Option

If you order the real-time clock option, the QTERM-R55 includes hardware support for the `date$` and `time$` qaBASIC commands.

3.7.3 Switching Regulator

The standard QTERM-R55 uses a switching regulator which allows the terminal to be powered from a 8-volt to 32-volt DC source.

3.8 QTERM Specifications

3.8.1 Environmental Characteristics

Environmental and power specifications are listed in Table 3-3.

Table 3-3. Environmental and Power Specifications

Parameter	Limits
Standard/Backlit Display usable temperature range	-10 to 60 °C
Wide- temperature Display usable temperature range	-20 to 70 °C
Storage temperature, all components	-40 to 85 °C
Maximum humidity range, all components	0 to 95%, non-condensing
Operating voltage range	8 to 32 volts DC
Battery Back Option Battery life Battery life, backlight on	 60 hours 24 hours

3.8.2 Electrical Characteristics

Typical current consumption is listed in Table 3-4.

Table 3-4. QTERM-R55 Current Consumption.

Version	Ireg (mA) @ 8V	Ireg (mA) @ 32V
EIA-232	88	33
Add for backlight	30	9
Add for each LED	4	2

APPENDIX A.

ASCII CHART

Least Significant Digit (hex)	Most Significant Digit (hex)							
	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

0345R3

NUL = blank
SOH = start of header
STX = start of text
ETX = end of text
EOT = end of transmission
ENQ = enquiry
ACK = acknowledge
BEL = bell
BS = backspace
HT = horizontal tab
LF = line feed
VT = vertical tab
FF = form feed
CR = carriage return
SO = shift out
SI = shift in
SP = space
DLE = data link escape
DC1 = device control 1 (XON)
DC2 = device control 2
DC3 = device control 3 (XOFF)
DC4 = device control 4
NAK = negative acknowledge
SYN = synchronization
ETB = end of text block
CAN = cancel
EM = end of medium
SUB = substitute
ESC = escape
FS = file separator
GS = group separator
RS = record separator
US = unit separator
DEL = delete/rubout

APPENDIX B.

QTERM-R55 CHARACTER CHART

The chart on the next page shows how the QTERM-R55 displays each of the 256 possible character values.

Where a dot pattern is shown, printing the corresponding code will cause the QTERM-R55 to display the dot pattern at the current cursor location.

Numbers in circles refer to these notes:

Œ These bytes are always ignored.

œ This is a space character

Note that, although the left half of this chart is similar to the ASCII chart in Appendix A, there are differences.

Least Significant Digit (hex)	Most Significant Digit (hex)															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	①	①	②	0	@	P	`	p	①	①	②	—	9	≡	α	p
1	①	XON	!	1	A	Q	a	q	①	①	｡	ｱ	ﾁ	ﾑ	ä	q
2	①	①	"	2	B	R	b	r	①	①	｢	ｲ	ﾂ	ﾒ	ß	θ
3	①	XOFF	#	3	C	S	c	s	①	①	｣	ｳ	ﾃ	ﾓ	ε	∞
4	①	①	$	4	D	T	d	t	①	①	､	ｴ	ﾄ	ﾔ	μ	Ω
5	①	①	%	5	E	U	e	u	①	①	･	ｵ	ﾅ	ﾕ	σ	ü
6	①	①	&	6	F	V	f	v	①	①	ｦ	ｶ	ﾆ	ﾖ	ρ	Σ
7	BEL	①	'	7	G	W	g	w	①	①	ｧ	ｷ	ﾇ	ﾗ	q	π
8	BS	①	(	8	H	X	h	x	①	①	ｨ	ｸ	ﾈ	ﾘ	ｷ	x
9	HT	①	)	9	I	Y	i	y	①	①	ｩ	ｹ	ﾉ	ﾙ	ｰ	ч
A	LF	①	*	:	J	Z	j	z	①	①	ｪ	ｺ	ﾊ	ﾚ	ｱ	千
B	LF	ESC	+	;	K	[	k	{	①	①	ｫ	ｻ	ﾋ	ﾛ	ｲ	万
C	LF	①	,	<	L	¥	l	|	①	②	ｬ	ｼ	ﾌ	ﾜ	ｳ	円
D	CR	①	-	=	M	]	m	}	①	②	ｭ	ｽ	ﾍ	ﾝ	ｴ	÷
E	①	①	.	>	N	^	n	→	①	②	ｮ	ｾ	ﾎ	ﾞ	ｵ	②
F	①	①	/	?	O	_	o	DEL	①	②	ｯ	ｿ	ﾏ	ﾟ	ö	■

1003R1

(Œ and œ are on the previous page.)

APPENDIX C.

QABASIC COMMAND SUMMARY

This appendix is an abbreviated summary of all of the available qaBASIC Commands. A more detailed descriptions of the commands are in Chapter 2.

Table C-1. qaBASIC Command Summary

Command	Description
<>	Conditional comparison - not equal
<=	Conditional comparison - less than or equal to
>=	Conditional comparison - greater than or equal to
=	Conditional comparison - equal to, or assignment
<	Conditional comparison - less than
>	Conditional comparison - greater than
x + y	Add two numbers
x - y	Subtract two numbers
x * y	Multiply two numbers
x / y	Divide two numbers
x ^ y	Take x to the y power
? x, ? string$	See Print
@	Can be used in place of at
:	Allows multiple commands to be placed on one line
;	Placed after a print statement to prevent a carriage return and linefeed from being appended to the end of the printed characters
#	An abbreviated form of REM - must be the first character on the line
'	An abbreviated form of REM
acos(x)	Calculates the ArcCosine of x
and	Used to combine to conditional statements
asc(string$)	Returns the ASCII value of the characture that is passed to it
asin(x)	Calculates the ArcSine of x
atan(x)	Calculates the ArcTangent of x
atan(y,x)	Calculates the ArcTangent of (y/x)
autoscroll	Used to control whether the display scrolls when the cursor is on the last line of the display and a linefeed is printed
autowrap	Used to control where the display will place the next character if the cursor is at the end of a line.

continued

Table C-1. qaBASIC Command Summary

Command	Description
backlight	Turns the backlight on or off
beep	Beeps for the number of seconds specified by x
bell	Same as beep
chr$(x)	Converts the ASCII value of x to a character
clear screen	Clears the display
close #x	Used to close file number x that was previously opened.
COM1	Reserved word - refers to the primary serial port
COM2	Reserved word - refers to the secondary serial port (if that option is available)
continue	Used in interrupt processing to deactivate an interrupt handling routine
contrast	Adjusts the display contrast
cos(x)	Calculates the Cosine of x
cursor	Used to change the appearance or location of the cursor
date$	Returns the date in a string
data	Used to define data to be read in using read
dim x	Sets the array size of x
else	Used with if, code following else will be run if the condition is false
end	Marks the end of program execution
endif	Marks the end of an if statement
eof	Used to determine if the end of a file has been reached
error	Used in an on error statement to implement non-fatal error handling
euler	The value of e^1
exp(x)	Calculates e^x
fi	Same functionality as endif
for	Used to perform a loop a certain number of times
frac(x)	Returns the decimal part of x
goint	Used to identify the label for the code used as an interrupt handler
gosub	Branches program execution to another part of the program
goto	Branches program execution to another part of the program
if	Used to perform conditional tests
inkey$	Gets a key from the keypad
input	Reads the users input
input at	Places the cursor at a specific location on the screen and receives input
instr(string$, chars$)	Searches string$ for chars$
int(x)	Returns the integer part of x
iskeypressed$	Used to get the key string for any key that may have been pressed - returns empty if no key has been pressed
keyclick	Turns the keyclick feature on or off

continued

Table C-1. qaBASIC Command Summary

Command	Description
keydef	Defines the keystring for a key
keyrelease	Used for enabling or disabling key release strings
label	Used to place a marker in the code for use with gotos and gosubs
led	Used to turn the terminal LEDs on or off
left$(string$,x)	Returns the left x characters from string$
len(string$)	Returns the length of string$
log(x)	Calculates the logarithim of x
lowbat	Used in the on lowbat statement that define a handler for the low battery interrupt
lower$(string$)	Converts the characters in string$ to lower case
ltrim$(string$)	Removes any spaces from the left side of the string
max(x,y)	Returns the higher of the x and y values
mid$(string$,x,y)	Returns the middle y characters from string$ starting at position x
min(x,y)	Returns the lower of the x and y values
mod(x,y)	Returns the remainder when dividing x by y
next x	Increments x and goes to the start of the for loop
normal	Used in the cursor normal statement to return the cursor mode to normal
not	Returns the inverse of whatever is after it
off	Used in other commands to turn something off
on	Used in other commands to turn something on
on x gosub	Performs a gosub to the label in the variable x
on x goto	Performs a goto to the label in the variable x
open #x	Open file x
or	Used to combine two conditional statements
pause x	Delays program execution for x seconds
peek(x$)	Used to read QTERM-R55 internal variables - legal values of x$ are: rows, columns numkeys, cursorX, cursorY, COM1, COM2, ostick
pi or PI	The value of pi 3.141592653589793 in 32 bit ANSI/IEEE 754-1985 format.
print	Prints whatever is after it to the display
print at	Works the same as print but outputs to a specified location
ran(x)	Returns a random number between 0 and x
read	Reads one data item at a time from the data lines in the program
recv	Receives input from the serial port and stores it as a number
recv$	Receives input from the serial port and stores it as a string
release	Used in a keydef statement to define a key release string
rem	Signals a comment; the line is not executed
restore	Used to specify the location that read gets its data from
return	Used at the end of a gosub to return to the next line in the main program flow

continued

Table C-1. qaBASIC Command Summary

Command	Description
right$(string$, x)	Returns the right x number of characters from string$
rtrim$(string$)	Removes the white spaces on the right side of the string$
seek #x to line y	Sets file pointer to line y in file x
seek #x y lines	Sets file pointer y lines away from current position
send	Sends data over the serial port
shift	Used in a keydef statement to define a shifted key string.
shiftstate	Sets the current state of the shift key
shutdown	Allows battery-powered terminals to be turned off by software - terminals not powered by a battery treat this command as an end
sin(x)	Calculates the Sine of x
step	Specifies the value to increment the variable in a for loop
sqrt(x)	Calculates the square root of x
str$()	Forces whatever is in the parenthesis to be viewed as a string
tan(x)	Calculates the Tangent of x
then	Placed at the end of an if statement to indicate where to start executing code
time$	Returns a string containing the current time
to	Used in a for loop to specify the upper bound of the loop
trim$(string$)	Removes spaces from both ends of a string
upper$(string$)	Converts the characters in string$ to upper case
val(string$)	Converts the string string$ to a number
wait x	Same as pause

x and y represent numbers
string$ represents a string

Other reserved keywords that currently do nothing but are not legal for variable names are:

arrow
break
bitmap
circle
dot
draw
erase
font
frame
interrupt
inverse
line
map
poke
printer
reverse
select
system
text
textwin
transparent
window
xmap
xtick
ymap
ytick

APPENDIX D.

USING THE R55 DOWNLOADER

D.1 Purpose

The R55 Downloader is a Windows® application that sends qaBASIC program files and QTERM-R55 firmware upgrades out a specified COM port on the PC.

D.2 Setup

To install the R55 Downloader on your PC hard drive, follow these steps:

- Copy the "R55 Downloader.exe" file from the diskette included with this manual to a directory on your hard drive.
- Test your installation by running the program. From the Start Menu in Windows select "Run." Type "x:\<directory>\R55 Downloader.exe" (including the quotes) where x: is the drive and <directory> is the directory where you copied the "R55 Downloader.exe" file. You may wish to create a shortcut to launch this program. Refer to your operating system documentation for details.
- A dialog box will appear with all the options that you need to configure the R55 Downloader to communicate with the QTERM-R55.

D.3 Basic Operation

D.3.1 Using the R55 Downloader

Launch the program as described above in Setup (section D.2). You may optionally specify the qaBASIC file that you wish to download on the command line (after “R55 Donwloader.exe”).

D.3.2 Selecting the file to download

Use the Filename field at the top of the dialog box to specify the name of the file to download. To browse for a file press the ... button. If a file with a .bin extension is entered the download type is automatically set to Firmware Upgrade. If a file with a .qbs or .bas extension is selected, the download type is automatically set to Application Download.

D.3.3 Selecting the serial port

Select the COM port which is attached to the QTERM-R55 from the COM Port box.

D.3.4 Setting the communication parameters

Select the baud rate, parity and flow control to match the communication settings of the primary COM port of the QTERM-R55.

It is possible to use the R55 Downloader with a QTERM-R55 that has been configured with EIA-485 on the primary interface. This requires a QCOM-4 serial interface device, available from QSI Corporation. Check the box labeled "This terminal is attached to a QCOM-4 using EIA-485 communications."

Other EIA-485 interface adapters may or may not work with the R55 Downloader. Contact QSI Techical Support.

D.3.5 Downloading the application

Prepare the QTERM-R55 to receive the application file or firmware upgrade via *Power-On Setup* [see Power-On Setup (section 1.1)]. Press the Download button to send the file to the terminal. A status bar will track the progress of the download. To stop sending data to the terminal, press Cancel. After the download is complete, the results of the download as well as any error messages will be displayed in the Download Results box.

D.3.6 Cancel

If the R55 Downloader is transmitting an application file or a firmware upgrade, this button terminates the transmission. At all other times, pressing this button closes the R55 Downloader.

D.4 Advanced Features

D.4.1 Port State

This Button indicates whether the serial port on the PC is open or closed. If the button is red, the port is closed. The button is green when the port is open. Other programs will not be able to access the serial port while R55 Downloader has it open. Normally the R55 Downloader opens the port automatically whenever it tries to send data to the serial port and closes the port whenever any communication parameters are changed. The port state may be toggled by pressing the button in the Port State box.

D.4.2 Send to R55

Enter text in this box and press the Transmit button. This feature is useful for sending short strings to the QTERM-R55 to test application programs.

D.4.3 Clear Box

Pressing this button erases the contents of the Download Results box.

D.4.4 Preprocessor Settings

The R55 Downloader contains a preprocessor for qaBASIC [see Preprocessor Directives (section D.5)]. The settings of this preprocessor can be changed with the Preprocessor Settings button. Pressing this button will bring up a dialog box listing the various options. Select "Preprocess application files before downloading" to control whether the preprocessor runs on an application file.

When the preprocessor runs, it parses all of the input files and generates an intermediate file; this intermediate file is then downloaded to the QTERM-R55. It is often useful to examine the contents of this intermediate file if the terminal reports that an application has errors. By default the intermediate file is R55.tmp.qbs and is stored in the directory which contains the application file. To change this, simply enter a new name in the "Save intermediate file as:" box, or browse using the ... button. To automatically delete the intermediate file when the R55 Downloader is closed, select "Delete intermediate file on exit." (Note: If you use more than one different intermediate file in one session, only the last file is deleted when the R55 Downloader is closed).

D.5 Preprocessor Directives

D.5.1 Overview

The R55 supports a few simple preprocessor directives. These directives are useful shortcuts to help speed application development. Each preprocessor directive is preceded with the '%' (percent) symbol. Each directive should be the last command on a line (i.e. don't try to use ':' to add more commands after the directive as they will be interpreted as part of the directive).

D.5.2 Include

Use the include directive to include the contents of another file at that location of the current file. The syntax is as follows:

```
%include "filename"
```

Example:

```
' Setup the custom keypad
%include "KeypadLayout.qbh"
```

In this example the contents of "KeypadLayout.qbh" are copied into the application file.

D.5.3 Define

The define directive is a macro substitution directive that redefines a symbol as something else. A symbol can be any string of characters. Symbols within quoted strings are not substituted. The syntax is:

```
%define <symbol> <definition>
```

Example:

```
' Shortcut to print out the serial port
%define PC1 print #COM1
```

The directive in this example will cause every occurrence of PC1 (except within quoted strings) to be replaced with "print #COM1"